ESTIMATION AND COMPENSATION OF SLIPPAGE BASED ON LASER-BASED ODOMETRY

A Dissertation
Presented to
The Academic Faculty

By

Bilal Ghader

In Partial Fulfillment
of the Requirements for the Degree
Master of Science in the
School of Electrical and Computer Engineering

Georgia Institute of Technology

December 2019

ESTIMATION AND COMPENSATION OF SLIPPAGE BASED ON LASER-BASED ODOMETRY

Approved by:

Dr. Cédric Pradalier
School of Computer Science
Georgia Institute of Technology

Dr. Patricio Vela
School of Electrical and Computer Engineering
Georgia Institute of Technology

Dr. Henri Owen
School of Electrical and Computer Engineering
Georgia Institute of Technology

Date Approved: December 5, 2019

”All models are wrong, but some are useful”

Georges Box

To my parents, to my brother, to my friends

Thank you

ACKNOWLEDGEMENTS

I would to thank Dr. Pradalier, the team at the DREAM Lab, and the administration at Georgia Tech Lorraine, and the Unité Mixte Internationale . This work would have not been accomplished without your support.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

SUMMARY

The project aimed to compensate for the slippage exhibited by a wheeled mobile robot in an off-road environment, in a trajectory following context. To do that, the odometry of the robot was calculated based on laser scan measurements. The slippage angle was estimated as an extra state using a state-estimator, the final controller was done using the back-stepping technique.

This manuscript, therefore, presents, the background of this project, and discusses the approach, the implementation details, and results of our work that was done on a skid-steer robot.

CHAPTER 1
INTRODUCTION AND BACKGROUND

1.1 Robots and Automation

Depending on who you ask, the definition of what is a robot varies a lot. The term robot can describe systems or machines with articulations similar to humans and be meant to mimic them. This type of robot is commonly known as humanoids. For example, NAO displayed in figure 1.1, a robot developed by SoftBank Robotics with educational purposes in mind. It can be also employed to describe arm-like systems used in factory-lines automation, such as the different solutions developed by the Kuka group seen in figure 1.2. The term is also employed to describe systems of the microscopic levels meant to assist in medical surgeries, kids' toys as seen in figure 1.3, or even traditional systems such as bicycles or cars deployed with an unusual level of autonomy or automation.

One might say that the only common point between all of these "robots" is a sensory-motor loop. Robots can be defined as a machine capable of accomplishing a certain level of tasks autonomously. To be able to do that, robots need to be able to gather information about the surrounding environment and process it accordingly.

1.2 Wheeled Mobile Robots

This thesis is interested in a certain subset of robots, commonly known as wheeled mobile robots. As their name suggests, are machines equipped with wheels and capable of moving autonomously in the surrounding environment. To be noted, several models of wheeled mobile robots exist, in this section, we will present a survey of the models relevant to our work.

Figure 1.1: NAO, the humanoid robots developed by SoftBanks

Figure 1.2: Robots employed in factory-line automation

Figure 1.3: A toy robot developed by Sphero in honor of BB-8 the Star Wars droid

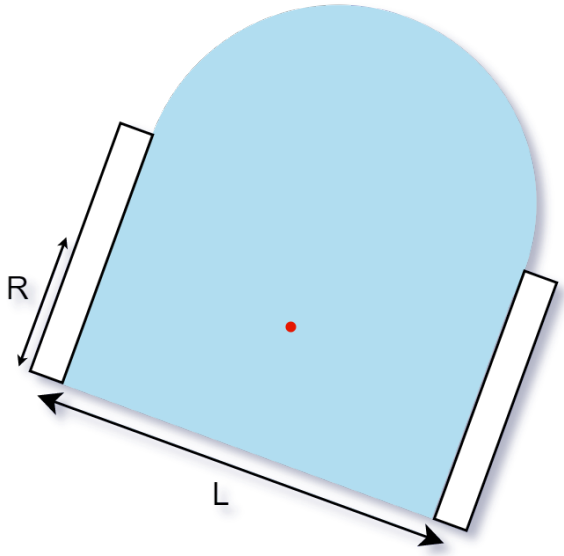


Figure 1.4: Differential drive robot schematic

1.2.1 Differential drive model

The differential drive model can be seen in figure 1.4. R is the radius of the wheel while L is the distance between the center of the two wheels. The red dot symbolizes the center of mass of the robot. The angle θ is the angle around the axis centered at the center of mass, it represents the orientation of the robot.

$$\begin{cases} \dot{X} = \frac{R}{2}(u_L + u_R)\cos\theta \\ \dot{Y} = \frac{R}{2}(u_L + u_R)\sin\theta \\ \dot{\theta} = \frac{R}{L}(u_L - u_R) \end{cases} \tag{1.1}$$

Equation 1.1 describes the dynamics of a differential drive robot, x and y are the coordinates of the center of mass and θ is the orientation of the robot. This model takes two inputs u_L and u_R, controlling the speed of left and right wheel respectively. The orientation of the system is controlled by the difference between u_L and u_R.

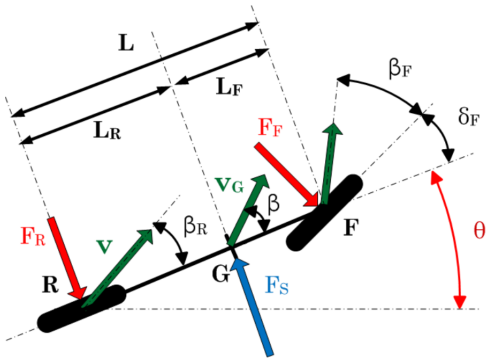


Figure 1.5: Bicycle model schematic

1.2.2 Bicycle model

As its name suggests, the bicycle model is inspired by a bicycle. A schematic of this model is presented in figure 1.5. In a similar manner to a bicycle, the robot is steered by controlling the front wheel orientation. The L presents the length of the body of the robot, while L_F and L_R are the distance between the center of the front and rear wheel respectively to the center of gravity of the robot G. β_R and β_F represents for the rear and front wheel respectively, the orientation difference between the velocity vector at the wheel and the orientation of the wheel. The angle θ is the angle with respect to the global frame coordinates. The movement of such a model is described by the following equation:

$$\begin{cases} \dot{X} = v\ \cos(\theta + \beta_R) \\ \dot{Y} = v\ \sin(\theta + \beta_R) \\ \dot{\theta} = v \cos(\beta_R) \frac{\tan(\delta_F + \beta_F) - \tan(\beta_r)}{L} \end{cases} \tag{1.2}$$

To be noted, this model can be extended to 4 wheels with 2 parallel sets of wheel. This would be a car model.

1.2.3 Skid-steer model

A Skid-steer robot is a robot with 4 wheels, operated independently. Unlike a bicycle robot, the skid steer robot does not benefit from a front steering axle. Instead, the skid steer rotation is controlled in a similar manner to the differential drive robot, by the difference in rotation speed between the left and right wheels.

The equation of motion of the Skid-Steered robot is presented in detail in [1]. A brief presentation of these equations are presented under two assumptions:

1. the center of mass and the geometric center of the body are the same;

2. the two wheels on the same side rotates at the same speed;

To express the model of the robot properly, we will define 2 inertial frames: The (X, Y) global frame and (x, y) local frame (defined with respect to the robot frame). [1]. In the local frame the linear velocity of the robot is expressed as $v = (v_x, v_y, 0)^T$, the angular velocity is $\omega = (0, 0, \omega_z)^T$. Let the vector $q = (X, Y, \theta)^T$ be the state vector defined in the global frame. the robot velocity can be described by the following state-space equation:

$$\begin{bmatrix} \dot{X} \\ \dot{Y} \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} \cos\theta & \text{-}\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \mathrm{v}_x \\ \mathrm{v}_y \\ \omega_z \end{bmatrix} \tag{1.3}$$

If the ω_i with $i = 1, 2, 3, 4$ describes the angular velocities of the four wheels denoted W_i, given our assumption we introduce ω_L and ω_R such as $\omega_L = \omega_1 = \omega_2$ and $\omega_R = \omega_3 = \omega_4$ as seen in figure 1.6

We denote the instantaneous center of rotations (ICR) of the left-side tread, right-side tread, with each 2 wheels on the same side being treated as a tread and the robot body as ICR_l, ICR_r, and ICR_G, with their respective (x, y) coordinates $(x_l, y_l), (x_r, y_r)$ and (x_G, y_G) these ICR lie on the line parallel to the x-axis [1]. which gives us the following

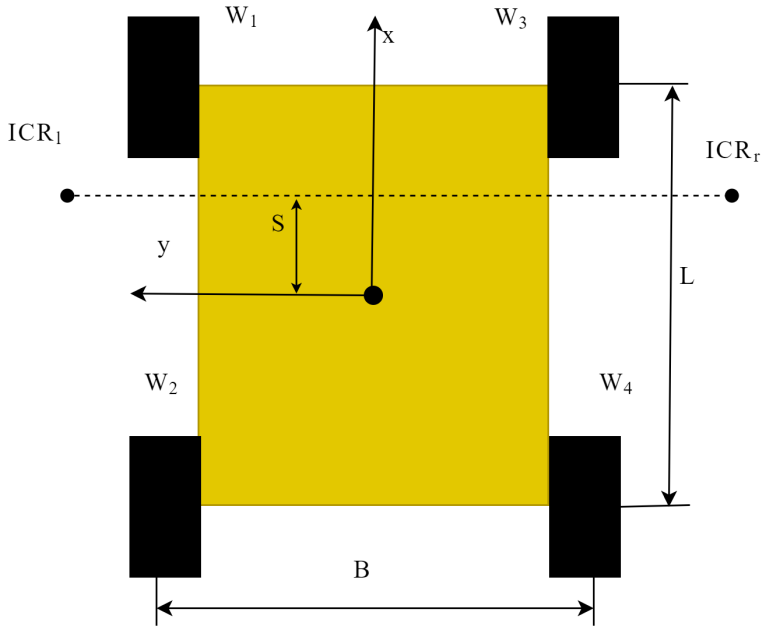


Figure 1.6: Figure displaying of a schematic of a skid-steering robots

relationships:

$$\begin{aligned} \mathrm{y}_g &= \frac{v_x}{\omega_z} \\ \mathrm{y}_l &= \frac{v_x \omega_l r}{\omega_z} \\ \mathrm{y}_r &= \frac{v_x \omega_r r}{\omega_z} \\ \mathrm{x}_G = x_l = x_r &= -\frac{v_y}{\omega_z} \end{aligned} \tag{1.4}$$

where r is the radius of the wheel. Finally, we can write the equation:

$$\begin{bmatrix} \mathrm{v}_x \\ \mathrm{v}_y \\ \omega_z \end{bmatrix} = \frac{1}{y_l - y_r} \begin{bmatrix} -\mathrm{y}_r & \mathrm{y}_l \\ \mathrm{x}_G & -\mathrm{x}_G \\ -1 & 1 \end{bmatrix} \begin{bmatrix} \omega_l r \\ \omega_r r \end{bmatrix} \tag{1.5}$$

Therefore, combining equations 1.3, 1.4, 1.5 can gives us a proper equation for the skid steer robot model. We can note it being a complex equation when compared to the differential drive and bicyle model equations.

To be noted, there is a certain parallel between skid-steer and differential drive robots.

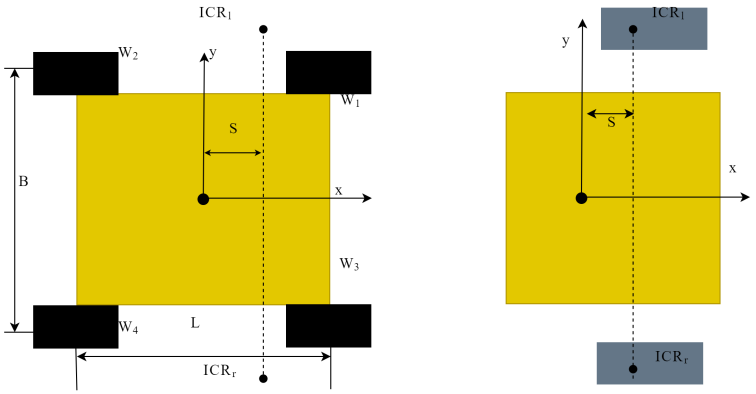


Figure 1.7: Equivalences between a skid-steer and a differential drive robot

As seen in figure 1.7. The main difference is that while the ICR of a differential drive robot is constant and located on point of contact with the ground.

1.3 Path and Trajectory Control

Now that the concepts of Wheeled Mobile Robots are explained, we will be presenting the problem at hand and the different attempted solutions in this section. Trajectory-following has always been a popular topic in the control community, early on considerable work was dedicated to this family of problems. Older references such as [2],[3], and [4] describes some of the early attempts to tackle this problem. The earlier approaches, however, did model the vehicle assuming ideal situations and not taking into consideration the effect of slippage or other non-linearities. To cope with this problem multiple control schemes were investigated, such as adaptive control [5], [6], neural networks [7] [8], and estimator based techniques.

Adaptive control model aims to control systems with unknown parameters, [5] and [6] showcases implementations of adaptive controllers for slippage compensation in trajectory following applications. In [5], the kinematic model of the unicycle robot was augmented to include the longitudinal slip through two slip parameters associated with the left and right wheels. A reference control input is calculated assuming no slippage. The slippage

parameters are considered constants, an adaptive control law is derived with respect to them. Simulation results showcase that this approach is capable of compensating for a constant slip. [6] builds upon this work by designing an adaptive control with respect to the slippage constant, using a different controller.

An alternative approach is to use neural networks to model uncertainty and non-linearities. The RBF (radial basis functions) network is well suited for such applications. An RBF is a two-layer network with a hidden layer and an output layer. In [8] and [7], RBF were employed to learn the non-linear dynamics of the system presents an implementation using RBF (radial-basis function) neural-network models. The RBF is a self-learning neural-network model, it is used to approximate the frictions dynamics. To guarantee the convergence of the system within an acceptable time period, a sliding mode controller is employed in [7].

The last approach that can be described is the one detailed in [9], [10]. In his work [9], Deremetz describes a possible algorithm to determine the slippage angle based on the movement of a robot along a predefined trajectory. The paper [10] describes how the angle of slippage can be estimated solely based on the input of the speed of the robot and its orientation as well as the predefined trajectory. This is done by expanding our state space by a state to model the slippage angle of the robot. Where β presents the slippage angle. Therefore, the equation of motion of the robots are expressed as:

$$\dot{\xi} = \begin{cases} \dot{\xi}_p = f(\xi_p, \xi_\beta) \\ \dot{\xi}_\beta = 0 \end{cases} \tag{1.6}$$

The states ξ are a combination of ξ_p the position states which is a function of position, angles β and of course their speeds. The state ξ_β which is the angles of slippage is assumed to be constant. But ξ_β is not constant, which is why our model is incorrect. To compensate for it, ξ_β is expressed as a function of $\tilde{\xi_p}$, where $\tilde{\xi_p}$ is the difference between ξ_p, the value calculated with our system dynamics model and the real states of the system.

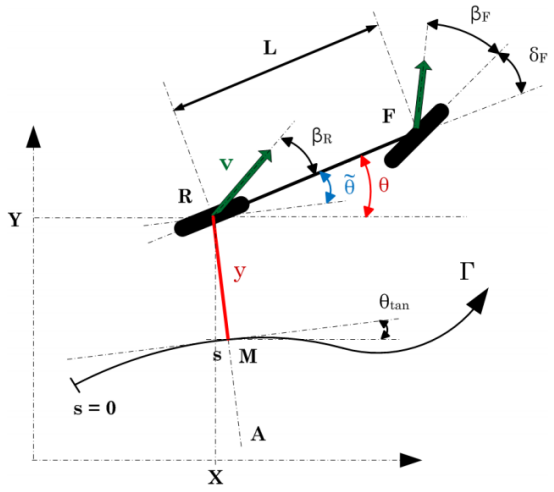


Figure 1.8: Reference fiugre used in the previous project

Finally, the dynamics of the estimated states would be:

$$\dot{\hat{\xi}} = \begin{cases} \dot{\hat{\xi}}_p = f(\xi_p, \hat{\xi}_\beta, v) + \alpha_{pos}(\tilde{\xi}_p) \\ \dot{\xi}_\beta = \alpha_\beta(\tilde{\xi}_p) \end{cases} \tag{1.7}$$

Where the α functions are non linear function meant to compensate the slippage.

1.4 Project Context

The project builds upon the work presented in [9] and [10], the entire project is presented in [11]. In the following section, we will be offering a summary of the approach used. The project aim was to create a trajectory following of a bicycle robot. In the figure 1.8, the trajectory Γ is the trajectory we aim to follow with a constant distance y. In order to follow the predefined trajectory Γ, we define the following states:

- s, given M as the closest point to R on the trajectory, s is the curvilinear distance on Γ of M. The curvature of Γ at M is $c(s)$.
- y, the lateral deviation between the robot and the trajectory Γ, it represents the algebraic distance between R and M.

- $\tilde{\theta}$, the angular deviation of the robot with respect to the trajectory Γ. It presents the angle between the absolute orientation of the robot, θ and the orientation of the tangent at the trajectory at M, noted θ_{tan}

The position of the rear wheel of the robot in the absolute coordinate system is denoted by X and Y

The kinematic equations corresponding to the extended model defined with respect to the trajectory are :

$$\begin{cases} \dot{s} = v\frac{\cos(\tilde{\theta}+\beta_R)}{1-c(s)y} \\ \dot{y} = v\sin(\tilde{\theta}+\beta_R) \\ \dot{\tilde{\theta}} = \dot{\theta} - \dot{\theta}_{tan} = v\cos(\beta_R)\frac{\tan(\delta_F+\beta_F)-\tan(\beta_R)}{L} - v\frac{c(s)\cos(\tilde{\theta}+\beta_R)}{1-c(s)y} \end{cases} \tag{1.8}$$

The dynamics equations are:

$$\begin{cases} \ddot{\theta} = \frac{1}{I_z}(L_R C_R \beta_R - L_F C_F \cos(\delta_F)) \\ \frac{d(v_G \sin\beta)}{dt} = \frac{C_F\beta_F(\delta_F)+C_R\beta_R}{m} - g\sin(\alpha) - v\cos(\beta_R)\dot{\theta} \end{cases} \tag{1.9}$$

With β and v_G being variables such as :

$$\begin{cases} \beta = \arctan\left(\frac{L_R\tan\beta_F+\delta_F+L_F\tanh\beta_R}{L}\right) \\ \mathrm{v}_G = \frac{v\cos\beta_R}{\cos\beta} \end{cases} \tag{1.10}$$

1.4.1 Observer formula

An state-observer was put in place using the kinematic model equations 1.2, we consider the following state vector ξ_{2WS}

$$\xi_{2WS} = \begin{bmatrix} \xi_{dev} \\ \xi_{\beta_i} \end{bmatrix} \tag{1.11}$$

The state vector is divided into two sub-states:

- $\xi_{dev} = [y \quad \tilde{\theta}]^T$ denoting the lateral and angular deviation of the robot with respect to the trajectory Γ.
- ξ_{β_i} with the deviation angles to be estimated.

The model of the evolution of the vector states is:

$$\dot{\xi} = \begin{bmatrix} \dot{\xi}_{dev} \\ \dot{\xi}_{\beta_i} \end{bmatrix} = \begin{bmatrix} f(\xi_{dev}, \xi_{\beta_i}, v, \delta_F) \\ 0_{2\times 1} \end{bmatrix} \tag{1.12}$$

Where the function $f(\xi_{dev}, \xi_{\beta_i}, v, \delta_F)$ is deduced from the equation 1.8. Since the angular deviation cannot be modeled, it is assumed to be 0. Hence we get the final equation of the observer:

$$\dot{\xi} = \begin{bmatrix} \dot{\xi}_{dev} \\ \dot{\xi}_{\beta_i} \end{bmatrix} = \begin{bmatrix} f(\xi_{dev}, \xi_{\beta_i}, v, \delta_F) + \alpha_{dev}(\tilde{\xi}_{dev}) \\ \alpha_{\beta_i}(\tilde{\xi}_{dev}) \end{bmatrix} \tag{1.13}$$

Where α_{dev} and α_{β_i} are two functions depending on the observations errors related to the robot equations. These functions have to be constructed in such a manner they allow the convergence of the state-vector estimation towards the real state-vector. The solution used is:

$$\begin{cases} \alpha_{dev}(\tilde{\xi}_{dev}) = \mathbf{K}_{dev}\tilde{\xi}_{dev} \\ \alpha_{\beta_i} = \mathbf{K}_{\beta}\left[\frac{\partial f}{\partial \xi_{\beta_i}}(\xi_{dev}, \hat{\xi}_{\beta_i}, v, \delta_F)\right]^T \tilde{\xi}_{dev} \end{cases} \tag{1.14}$$

where:

- $\mathbf{K}_{dev}$ is a 2x2 diagonal matrix,
- $\mathbf{K}_{\beta}$ is a positive scalar.

The equation 1.12 covers only the kinematic model of the robot, it can be expanded to cover the dynamical model of the robot expressed in 1.9:

$$\dot{\xi} = \begin{bmatrix} \dot{\xi}_{dev} \\ \dot{\xi}_{dyn} \\ \dot{\xi}_{\beta_i} \\ \dot{\xi}_{C_i} \end{bmatrix} = \begin{bmatrix} f(\xi_{dev}, \xi_{\beta_i}, v, \delta_F) \\ g(\xi_{dyn}, \xi_{\beta i}, \delta_F) \\ 0_{2\times 1} \\ 0_{2\times 1} \end{bmatrix} \tag{1.15}$$

With $\xi_{dyn} = [\theta \quad (v_G \sin\beta)]^T$ and $\xi_{Ci} = [C_F \quad C_R]^T$ where C_F and C_R are the rigidity of drift of the front and back wheel.

The Final equation would be :

$$\dot{\xi} = \begin{bmatrix} \dot{\hat{\xi}}_{dev} \\ \dot{\hat{\xi}}_{dyn} \\ \dot{\hat{\xi}}_{\beta_i} \\ \dot{\hat{\xi}}_{C_i} \end{bmatrix} = \begin{bmatrix} f(\xi_{dev}, \xi_{\beta_i}, v, \delta_F) + \alpha_{dev}(\tilde{\xi}_{dev}) \\ g(\xi_{dyn}, \xi_{\beta i}, \xi_{Ci}, v, \delta_F) + \alpha_{dyn}(\tilde{\xi}_{dyn}) \\ \alpha_{\beta_i}(\tilde{\xi}_{dyn}, \tilde{\xi}_{dev}) \\ \alpha_{C_i}(\tilde{\xi}_{dyn}) \end{bmatrix} \tag{1.16}$$

The equation of the αs functions would be:

$$\begin{cases} \alpha_{dev}(\tilde{\xi}_{dev}) = \mathbf{K}_{dev}\tilde{\xi}_{dev} \\ \alpha_{dyn}(\tilde{\xi}_{dyn}) = \mathbf{K}_{dyn}\tilde{\xi}_{dyn} \\ \alpha_{\beta_i}(\xi_{dev}, \xi_{dyn}) = \frac{\mathbf{K}_\beta}{\mathbf{K}_\alpha}\left[\frac{\partial f}{\partial \xi_{\beta_i}}(\xi_{dev}, \hat{\xi}_{\beta_i}, v, \delta_F)\right]^T \tilde{\xi}_{dev} + \frac{\mathbf{K}_C}{\mathbf{K}_\alpha}\left[\frac{\partial g}{\partial \xi_{\beta_i}}(\xi_{dev}, \hat{\xi}_{\beta_i}, \hat{\xi}_{C_i}, v, \delta_F)\right]^T \tilde{\xi}_{dyn} \\ \alpha_{C_i}(\tilde{\xi}_{dyn}) = \mathbf{K}_C\left[\frac{\partial g}{\partial \xi_{\beta_i}}(\xi_{dev}, \hat{\xi}_{\beta_i}, \hat{\xi}_{C_i}, v, \delta_F)\right]^T \tilde{\xi}_{dyn} \end{cases} \tag{1.17}$$

In equation 1.17, $\mathbf{K}_{dev}$ and $\mathbf{K}_{dyn}$ are each a 2×2 positive diagonal matrices and $\mathbf{K}_\beta$,$\mathbf{K}_C$, and $\mathbf{K}_\alpha$ are positive scalar. K_α is just meant to harmonise K_C and K_β since they do not share the same unit and scale.The function $f(\xi_{dev}, \xi_{\beta_i}, v, \delta_F)$ and $g(\xi_{dyn}, \xi_{\beta i}, \xi_{Ci}, v, \delta_F)$ are deduced directly from equation 1.2 and 1.9.

CHAPTER 2

TECHNICAL APPROACH

In this chapter we will be discussing how the approach described earlier was adopted in our project, figure 2.1 showcases a high-level description of this approach. The state estimator takes as input the sensor signal from the robot. Several alternatives can be used for this block. In this implementation, the states of the system are estimated using a laser scan matcher, a tool that takes successive scans and estimates the movement of the robot.

The estimation of slippage block can, therefore, be either the regular state-estimator as described in [9],[10] or a neural network estimation of the value of the slippage.

Finally, the controller block uses the knowledge produced of the system and knowledge of the trajectory to create the twist signal sent to the robot.

The goal trajectory is published as a function of time, it is useful to approximate the slippage by computing the difference between the current location and the calculated location according to the known dynamics.

Figure 2.2 presents a typical software architecture of an autonomous mobile robot. The sensor interface corresponds to the hardware sensors installed on the robot and the raw data they output, which are labeled sensor data in this figure. The perception blocks correspond to all the operations and calculations performed on these data, these operations can be

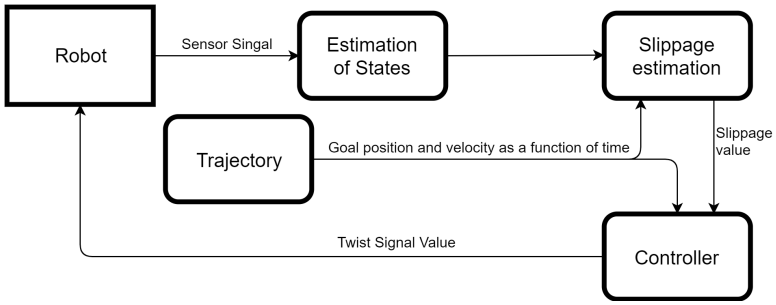


Figure 2.1: A high level description of the implemented approach

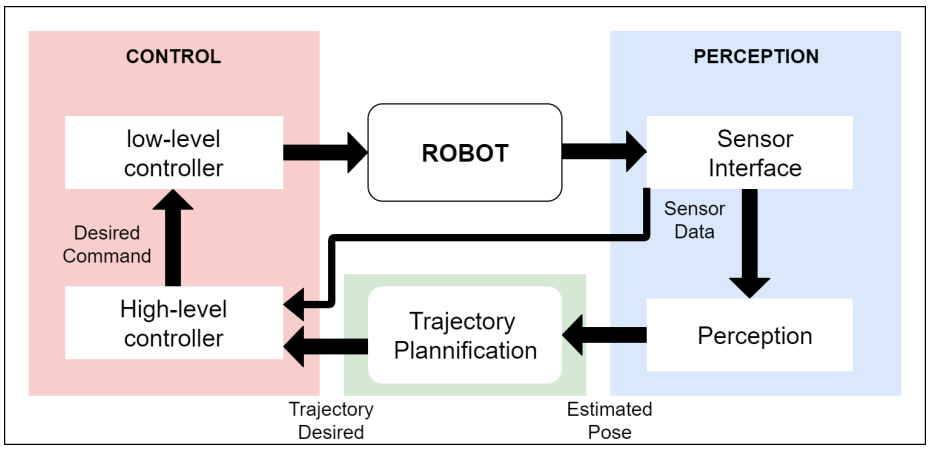


Figure 2.2: Typical software architecture of an autonomous mobile robot

simple signal processing or more complex data processing algorithms like the one used in our solution. It is also important to differentiate high-level from the low-level controller. In a similar manner to the nomenclature in the programming language, high-level control is an abstract one, in our case, the high-level control would be the twist command (a velocity and rotation command) sent to the robot. The low-level control would be the individual rotation speed given to each of the wheels. In this project, low-level control was provided by the ROS stacks and not by the user.

2.1 Robot model adopted

The entire project was being tested on Clearpath Husky robot is seen in figure 2.4. The project was implemented and tested using the ROS platform, the project aimed at only implementing a high-level controller of the robot and not controllers on the level of individual wheels. The robot is a skid-steer robot, however, given the advantage we have of being able to deal with the high-level controller of the robot, it was decided to model the robot as a differential drive one. To be noted, the difference between the two models was accounted for by the adaptive controller which dealt with it as an extra slippage.

Figure 2.3 showcases a schematic of the model we adapted,(x, y, θ) presents the position and orientation of the robot. The y in red showcases the lateral deviation of the robot

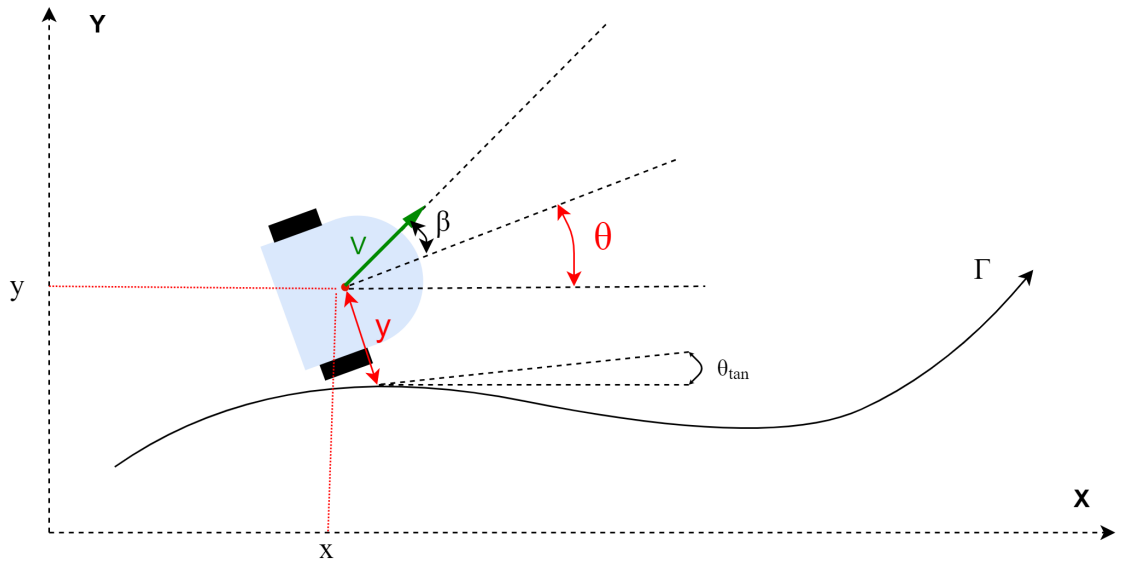


Figure 2.3: Schematic figure showing a unicyle robot in a trajectory following context

with respect to the trajectory of Γ. θ_{tan} is the deviation between the robot orientation and the orientation desired at the closed point in the path (presented by the tangent to the path at that point). v showcases the real velocity vector of the robot movement, β is the angle between the robot orientation and the orientation of this velocity vector (the difference is due to the slippage we are trying to compensate).

The adopted equation 1.8 is:

$$\begin{cases} \dot{s} = v\frac{\cos(\tilde{\theta}+\beta)}{1-c(s)y} \\ \dot{y} = v\sin(\tilde{\theta}+\beta) \\ \dot{\tilde{\theta}} = \dot{\theta} - \dot{\theta}_{tan} = v\cos(\beta) - v\frac{c(s)\cos(\tilde{\theta}+\beta)}{1-c(s)y} \end{cases} \tag{2.1}$$

2.2 Extroceptive Sensor

An exteroceptive sensor is a sensor that measures stimuli external to the robot, which makes its capacity to estimate the movement of the robot independent of any internal knowledge. Given our aim to approximate the movement of the robot properly, an exteroceptive sensor

Figure 2.4: An image of the Husky robot used for testing

had to be used. Several choices were possible, we will be describing them in this section and then explain the decision we made.

2.2.1 Cameras and Visual Odometry

One of the most obvious sensors in this situation is the camera. Many stacks do exist too, approximate the movement based on a sequence of a picture taken by a camera such as the solution presented in [12].

However, it is important to note that the monocular camera setup is incapable of recover the entire 3D structure of the world, the only possible reconstruction is a scale-less reconstruction of the movement. But, this problem can be resolved by using a stereo-camera setup, allowing it to recover the scale. Another solution that can be used is an RGBD, a camera capable of capturing the depth in addition to the regular Red, Green, and Blue arrays.

We decided however not to use a camera setup for our solution for several reasons:

- An image-to-image matching algorithm is very computationally expensive and may add a delay in the process which does not allow us to deploy it in real-time.

- We aimed to test the solution outdoor, image-to-image matching solutions are very sensitive towards lighting conditions which are problematic in a natural environment.

2.2.2 Laser Scan sensors

Another type of exteroceptive sensors are laser scanners, these sensors project an array of laser beams, and measures the reflection of these beams. This operation allows it to have a PointCloud describing the objects around the sensor, or rather their distance to the sensor. We discern between two types of laser scan sensors:

2D Laser

The 2D laser scan sensors project only a one-dimensional array, the recovered PointCloud scan would, therefore, be a 2-dimensional one. In 3D, the presented point cloud would present a section of the world around the robot.

3D Laser

The 3D laser scanner project a 2-dimensional array, the recovered PointCloud scan would, therefore, be a 3-dimensional one. Of course, this type is much more robust. However, a 3D laser scanner is more expensive and unfortunately, we did not have one available in the laboratory. Thus, we decided to use a 2D laser scan.

2.2.3 Different Laser Scan Solution Evaluation

In our case, we decided to go with a 2D laser because of availability. To recover the movement of the robot, a scan matcher stack is needed. A scan matcher is a software capable of taking in successive measurements of a Laser Scan and estimate the movement of the robot.

We assessed two different scan matching toolkit to be used to recover the odometry of the robot:

Characteristic	Norlab stack	LaserScan Matcher Stack
built-in odometry support	yes	no
built-in first guess support	no	yes

Table 2.1: Comparaison between the two scan matcher solution

Norlab ICP mapper

The Norlab ICP mapper presented in [13] [14] is a tool that takes in a point cloud and provides a map of the world around them laser, as well as the movement of the laser in that world. Hence it would allow us to calculate the odometry of the robot based on the successive laser scans.

The Norlab solution, however, lacked the capacity of having a first-guess input, a feature offered by the Laser scan matcher in a straight forward manner.

Laser Scan matcher

A laser scan matcher is a tool provided by the scan tools package [15] [16], a package that encompasses several tools meant for laser scan processing. The scan matcher, in particular, allows the recovery of the movement of the robot starting with the laser scan of it. However, unlike the Norlab implementation, this implementation only outputs the pose of the robot. Hence, if it is to be used it should be modified to support it.

However, unlike the Norlab version, the laser scan matcher as a tool that can take the odometry computed by the robot as its first guess in a straightforward manner, allowing us to have a decent first guess. Hence, we decided to used the laser scan matcher package provided by the scan tools kit.

2.3 Dynamics Observer

In order to observe the dynamics of the system, the same observer principle described in equations 1.15 and 1.16 was employed. The modified version of these equation: the robots

are expressed as:

$$\dot{\xi} = \begin{bmatrix} \dot{\hat{\xi}}_{dev} \\ \dot{\hat{\xi}}_{dyn} \\ \dot{\hat{\xi}}_{\beta} \\ \dot{\hat{\xi}}_{C} \end{bmatrix} = \begin{bmatrix} f(\xi_{dev}, \xi_{\beta}, v) \\ g(\xi_{dyn}, \xi_{\beta}, \xi_{C}, v) \\ 0 \\ 0 \end{bmatrix} \tag{2.2}$$

As noted in equation 2.2, The states ξ are a combination of ξ_{dev}, the position states which is a function of position, the angle β and of course their velocities and xi_{dyn} that includes the dynamics state of the system. The states ξ_{β} and ξ_{C} are now a one-dimensional vector. This is due to the model change since we are modeling the robot as a differential drive one, we only have one slippage angle. As seen in figure 2.3 In an ideal case, there is no slippage, hence the state ξ_{β} and ξ_{C} are assumed to be 0. But ξ_{β} is not null, which is why our model is incorrect. In a similar manner to previously presented approach, to compensate for the error, ξ_{β} is expressed as a function of $\tilde{\xi}_{dev}$, where $\tilde{\xi}_{p}$ is the difference between ξ_{p}, the value calculated with our system dynamics model and the real states of the system.

Hence, the equation 2.3 that expresses a correct modeling of the states:

$$\dot{\xi} = \begin{bmatrix} \dot{\hat{\xi}}_{dev} \\ \dot{\hat{\xi}}_{dyn} \\ \dot{\hat{\xi}}_{\beta} \\ \dot{\hat{\xi}}_{C} \end{bmatrix} = \begin{bmatrix} f(\xi_{dev}, \xi_{\beta}, v) + \alpha_{dev}(\tilde{\xi}_{dev}) \\ g(\xi_{dyn}, \xi_{\beta}, \xi_{C}, v) + \alpha_{dyn}(\tilde{\xi}_{dyn}) \\ \alpha_{\beta}(\tilde{\xi}_{dyn}, \tilde{\xi}_{dev}) \\ \alpha_{C_i}(\tilde{\xi}_{dyn}) \end{bmatrix} \tag{2.3}$$

Where theα functions are non linear function meant to compensate the slippage. The equation of the αs functions would be:

$$\begin{cases} \alpha_{dev}(\tilde{\xi}_{dev}) = \mathbf{K}_{dev}\tilde{\xi}_{dev} \\ \alpha_{dyn}(\tilde{\xi}_{dyn}) = \mathbf{K}_{dyn}\tilde{\xi}_{dyn} \\ \alpha_{\beta}(\xi_{dev}, \xi_{dyn}) = \frac{\mathbf{K}_\beta}{\mathbf{K}_\alpha}\left[\frac{\partial f}{\partial \xi_\beta}(\xi_{dev}, \hat{\xi}_\beta, v)\right]^T \tilde{\xi}_{dev} + \frac{\mathbf{K}_C}{\mathbf{K}_\alpha}\left[\frac{\partial g}{\partial \xi_\beta}(\xi_{dev}, \hat{\xi}_\beta, \hat{\xi}_C, v)\right]^T \tilde{\xi}_{dyn} \\ \alpha_{C_i}(\tilde{\xi}_{dyn}) = \mathbf{K}_C\left[\frac{\partial g}{\partial \xi_{\beta_i}}(\xi_{dev}, \hat{\xi}_\beta, \hat{\xi}_C, v)\right]^T \tilde{\xi}_{dyn} \end{cases} \tag{2.4}$$

The implementation of all of this can be seen in details in the algorithm 1

Algorithm 1 The Observer algorithm

global variables: $\hat{y}, \hat{\tilde{\theta}}, \beta, \hat{\beta}$

Observer Estimation
function INTIALIZATION($y, \tilde{\theta}, \dot{\theta}$)
    $\hat{y} \leftarrow y, \hat{\tilde{\theta}} \leftarrow \tilde{\theta}, \hat{\dot{\theta}} \leftarrow \dot{\theta}$
    $V_g\beta \leftarrow 0, \hat{\beta} \leftarrow 0, \hat{C} = C_{init}$
end function
function COMPUTATION OF ESTIMATION ERROR
    $\tilde{y} = y - \hat{y}$
    $\tilde{\tilde{\theta}} = \tilde{\theta} - \hat{\tilde{\theta}}$
    $v_G\beta - \hat{v_G\beta}$
end function
function COMPUTATION OF DERIVATIVES ESTIMATES($v, \dot{\theta}, \tilde{\theta}, a$)
    $\hat{\dot{y}} = v\sin\tilde{\theta} + \hat{\beta} + K_{dev}\tilde{y}$
    $\hat{\dot{\tilde{\theta}}} = K_{dev} * \tilde{\tilde{\theta}}$
    $\hat{\ddot{\theta}} = K_{dyn} * \tilde{\dot{\theta}}$
    $\hat{\dot{V}_{g\beta}} = \frac{\hat{C}}{\beta} - v\cos\beta\dot{\theta} + K_{dyn}v_{g\tilde{\beta}}$
    $\dot{\hat{\beta}} = \frac{K_\beta}{K_a}(v\cos\beta + \tilde{\theta}) + \frac{K_C}{K_a}(\frac{\hat{C}}{m} + v\sin\beta + \dot{\theta}\tilde{V_g}\beta)$
    $\dot{\hat{C}} = K_C\frac{\beta}{m}\tilde{v}_g\beta$
end function
function COMPUTATION OF THE ESTIMATED VARIABLE
    $\hat{y} \leftarrow \hat{\dot{y}}\Delta T$ ▷ ΔT is the sampling period
    $\hat{\tilde{\theta}} \leftarrow \hat{\tilde{\theta}} + \hat{\dot{\tilde{\theta}}}\Delta T$
    $\hat{v_G\beta} \leftarrow v_G\beta + \dot{v_G\beta}\Delta T$
    $\hat{\beta} = \hat{\beta} + \dot{\hat{\beta}}\Delta T$
    $\hat{C} = \hat{C} + \dot{\hat{C}}\Delta T$
end function

2.4 Slippage compensation control

In this section, a detailed explanation of the control scheme implemented is presented. Given the theory provided in previous equations, the robot position and slippage angle can be estimated robustly, allowing us to estimate also its deviation with respect to the desired trajectory.

Given all this knowledge, it is possible to implement a control law for the twist of the robot that takes into consideration all of these factors to provide a satisfactory trajectory following. Hence a command law was synthesized using Backstepping control. This section will explain the control law used with the first subsection providing a review of Backstepping control, and the second subsection describing the implementation used.

2.4.1 Review of the control by backstepping technique

In this subsection, we will present a brief review of control by backstepping. A more detailed explanation can be found in [17].

Assume a system of the form:

$$\begin{aligned} \dot{X} &= F(X) + G(X)x \\ \dot{x} &= f(X,x) + g(X,x)u \end{aligned} \tag{2.5}$$

Where $X \in \mathbb{R}^n$ and $x \in \mathbb{R}$ and $g(X,x) \neq 0 \quad \forall \quad (X,x) \in \mathbb{R}^{n+1}$.

Suppose a stabilizing feedback $u = \alpha(X)$, $\alpha(0) = 0$ is available for $\dot{X} = F(X) + G(X)x, F(0) = 0$ With a Lyapunov function V such that :

$$\frac{\partial V}{\partial X}(F(X) + G(X)\alpha(X)) \leq -W(X) < 0 \quad \forall X \neq 0$$

and use the preliminary feedback

$$u = \frac{1}{g(X,x)}(-f(X,x) + v) \tag{2.6}$$

The subsystem becomes a pure integrator $\dot{x} = v$. Let modify $\alpha(X)$ to stabilize our system. Define the error variable $z = u - \alpha(X)$ and the change of variables: $(X, x) \longrightarrow (X, z)$.

In order to control our system we replace v by:

$$v = \dot{\alpha} - \frac{\partial V}{\partial X}G(X) - \mathbf{K}z \quad z \triangleq x - \alpha(X) \quad \mathbf{K} > 0 \tag{2.7}$$

The final u control command will be:

$$u = \frac{1}{g(X,x)}\left(-f(X,x) + \dot{\alpha} - \frac{\partial V}{\partial X}G(X) - \mathbf{K}z\right) \tag{2.8}$$

To be noted, this method can be applied recursively to systems of the form:

$$\begin{aligned} \dot{x}_1 &= f_1(x_1) + g_1(x_1)x_2 \\ \dot{x}_2 &= f_2(x_1, x_2) + g_2(x_1, x_2)x_3 \\ &\vdots \\ \dot{x}_n &= f_n(x) + g_n(x)u \end{aligned} \tag{2.9}$$

where $g_i(x_1, \ldots, x_i) \neq 0 \quad \forall x \in \mathbb{R}^n, i = 1, 2, 3, \ldots, n$

2.4.2 Controller implementation

The principles discussed in the previous section are described in the previous subsection are used to create a cascaded controller. Let y be the lateral deviation of the robot, and let

e_y be the lateral deviation error.

$$\begin{aligned} e_y &= y \\ \dot{e}_y &= \dot{y} \\ &= \mathbf{K}_y e_y \end{aligned} \tag{2.10}$$

Where $\mathbf{K}_y$ is negative scalar. Replacing the y by this in the equation 2.1 would give:

$$v \ \sin(\tilde{\theta} + \hat{\beta}) = k_y e_y \tag{2.11}$$

Looking at $\tilde{\theta}$ as an intermediary state, let $u_{\tilde{\theta}}$ be the desired $\tilde{\theta}$; $u_{\tilde{\theta}}$ can be chosen as:

$$\mathbf{u}_{\tilde{\theta}} = \arcsin\left(\frac{\mathbf{k}_y(y)(v)}{\hat{\beta}}\right) \tag{2.12}$$

The equation 2.12 guarantee the convergence of $\tilde{\theta}$ to 0. Hence, the new error we have to deal with is :

$$e_{\tilde{\theta}} = \tilde{\theta} - u_{\tilde{\theta}} \tag{2.13}$$

In a similar manner, the error dynamics can be posed as:

$$\dot{e}_{\tilde{\theta}} = k_{\tilde{\theta}} e_{\tilde{\theta}} \tag{2.14}$$

with $k_{\tilde{\theta}}$ being a negative scalar. To be noted, the response time of this step should be slower than the gain controlled by the first one. Hence, we can assume that $u_{\tilde{\theta}}$ is invariant with respect to time. The expression of the derivative of angular deviation is, therefore:

$$\dot{e}_{\tilde{\theta}} = \dot{\tilde{\theta}} = \dot{\theta} - \dot{\theta}_{tan} = \mathbf{k}_{\tilde{\theta}} e_{\tilde{\theta}} \tag{2.15}$$

The goal $\dot{\theta}$ is therefore:

$$u_{\dot{\theta}} = k_{\tilde{\theta}}(\tilde{\theta} - u_{\tilde{\theta}}) + \frac{v\, c(s)\, cos(\tilde{\theta} + \beta)}{1 - c(s)y} \tag{2.16}$$

In addition to the angular controller presented, the controller includes a linear controller:

$$\dot{x} = \dot{x}_{ref} + K_x \tilde{x} \tag{2.17}$$

where K_x is a negative constant, and $\tilde{x}$ is the linear error, and $\dot{x}_{ref}$ being referenced linear velocity as given by the reference path.

2.4.3 Augmented controller

During the testing phase, it was noted that the controller based only on the estimator results have jittery results. This is due to two reasons, the robot used is a skid-steering robot hence its response to angular velocity exhibits a dead-zone below a certain value. Additionally, the estimator is by design in delay with respect to the system and hence the compensation scheme is delayed with respect to the robot dynamics. This is not problematic in [10] because of the robot used is a bicycle robot. The approach presented in the previous subsection does not take advantage of the direct knowledge of the path and the lateral error. This creates a delay in the controller. To compensate this, a feed-forward component was added to the controller:

$$\dot{\theta}_{ff} = \dot{\theta}_{ref} + K_{angular}\tilde{\theta} + K_y x_{ref} * \exp(-\tilde{\theta}^2)\tilde{y} \tag{2.18}$$

This feed-forward controller command is a slightly modified version of the trajectory-following controller proposed in [18] using the $\exp(-x^2)$ instead of $\mathrm{sinc}(x)$. However, in [18] it was employed assuming no slippage was happening.

The final command of $\dot{\theta}$ is therefore a weighted summation of 2.16 and 2.18, with the 2.16 having a higher weight.

Given the feed-forward term, the system can mitigate the delay problem and that will be discussed in the results section. The detailed implementation of this can be seen in the algorithm 2.

Algorithm 2 The Controller algorithm

$\mathbf{u}_{\tilde{\theta}} \leftarrow \arcsin\left(\frac{\mathbf{k}_y(y)(v)}{\hat{\beta}}\right)$

$\dot{\theta}_{ff} \leftarrow \dot{\theta}_{ref} + K_{angular}\tilde{\theta} + K_y x_{ref} * exp(-\tilde{\theta}^2)\tilde{y}$

▷ The feed-forward components of the controller

$u_{\dot{\theta}} \leftarrow k_{\tilde{\theta}}(\tilde{\theta} - u_{\tilde{\theta}}) + \frac{v\, c(s)\, cos(\tilde{\theta}+\beta)}{1-c(s)y}$ ▷ intermediary value

$u_{\dot{\theta}} = \mathbf{sat}(u_{\dot{\theta}} + \dot{\theta}_{ff})$ ▷ **sat** is the saturation function

CHAPTER 3
EVALUATION METHODOLOGY

This Chapter will concern itself with the evaluation methodology used throughout the project, it will introduce the different values of interests, how they are measured, and what they mean.

3.1 Measurements and values of interest

Figure 2.1 showcases the different tools implemented in this project. Implementation-wise each of the following tool is independent of the other to allow to tweak / calibrate it independently:

- The Matcher tool: responsible for taking in recorded or live laser scan messages and provide us with an estimation of the odometry. It was calibrated with pre-recorded messages.
- The observer: Using the estimation provided by the Matcher, the observer's goal is to provide the lateral and angular deviation.
- The controller: Responsible of using the estimated data and send the control signal to the robot as seen in subsections2.4.1, 2.4.3.

We will be detailing all the values of interest, and how each of these items was calibrated.

To evaluate our system, we record and analyze different values, the most important of which are:

- y, the lateral deviation, represents the lateral distance between the robot and the trajectory followed. It can be seen in red in 2.3.

- $\tilde{\theta}$, the angular deviation, represents the orientation difference between the desired orientation and the real orientation θ.
- The position of the robot is (x, y) the 2-dimensional space.

3.2 Evaluation of the Laser Scan tool solutions

It was already explained that there are several possible solutions to the 2D laser scan matcher. The capabilities of these tools to provide decent odometry and be robust were to be assessed. Therefore, we started by recording the laser scan messages based on random drives of the robot in the natural environment. These recordings were stored in a rosbag format to be played back later.

Later on, with the help of simulation and visualization tools provided by the ROS stack, these recordings were replayed with the matcher implementation(s) over them. Several runs of this step were performed with the modification of the different parameters in the matcher(s) to get a satisfactory solution.

To be noted, the Laser Scan matcher implementation was modified to support publishing odometry messages, the same process was repeated to make sure the new implementation works properly.

In a second step, we were concerned with what kind of accuracy can these implementations provide us. Of course, we cannot evaluate that in a real-life scenario since we do not have a ground-truth to rely on. Therefore, a simulation setup (detailed in section 4) was made. This allowed us to obtain a ground truth of the position of the robot that was provided by the environment tool. This is one of the main assumptions of the project, the solution implemented can give a decent performance if a good matcher performance is guaranteed.

A decent performance of the scan matcher is performance comparable to the results of an RTK-GPS. Therefore, we aim to get a position estimate with less than 10 cm error. To evaluate this, we compared the results of the laser scan position estimates to the ground

truth of the position. Of course, this was only done in the simulation since obtaining a ground truth in real-life in impossible.

3.3 Calibration of the observer and controller

A badly calibrated observer is either very fast of very slow. If it was too fast, the estimation of the state would tend to grow exponentially and reach unrealistic value (angular deviation superior to $\frac{\pi}{2}$ for example). A set of pre-recorded messages were used to make sure the observer estimation moves at a reasonable pace.

In a second step, we were concerned with what kind of accuracy can these implementations provide us. Of course, we cannot evaluate that in a real-life scenario since we do not have a ground-truth to rely on. Therefore, a simulation setup (detailed in section 4) was made. This allowed us to obtain a ground truth of the position of the robot that was provided by the environment tool. This is one of the main assumptions of the project, the solution implemented can give a decent performance if a good matcher performance is guaranteed. The controller is the last layer of the implementation. Similarly, the controller calibration was an iterative process. However, here the first step was directly the simulations. After several tests on the simulations, the controller was deployed on the real robot and evaluated.

Of course, the value of the controller was capped to make sure the robot does not receive unrealistic values as seen in 2.

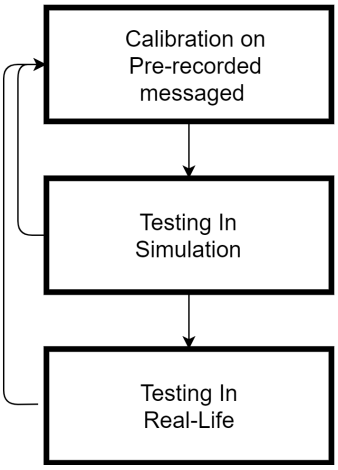


Figure 3.1: Schematic showcasing the step of calibration of the controller and observer

CHAPTER 4
TESTING AND RESULTS

The following section will be discussing the different steps in the validation process during the simulation and testing of the project. We will start by discussing the simulations performed throughout the project. Later on, we will be discussing the live tests performed to validate our work.

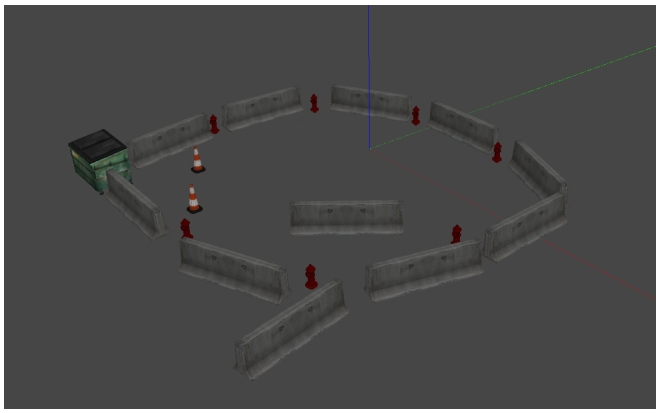
4.1 Simulation Results

Starting with the simulation, in fact, we performed several tests in simulation. The tests performed aimed at comparing the different laser scan odometry tools, validating the accuracy of the laser scan odometry results with respect to the ground truth, and testing the entire system.

4.1.1 simulation environments and tools

The entire simulation was performed using the Gazebo tool. The path was published as a succession of position, orientation, and velocity. The path published $p(t)$ is the desired position and orientation of the robot at the time t. A simulated map was built as a gazebo virtual world that can be seen in figure 4.1 . A tool was created to publish the target position of the robot.

To simulate the robot, we used to husky gazebo robot package built for that purpose. We enabled the support of the laser scan package to perform our simulations. The simulated robot can be seen in figure 4.2



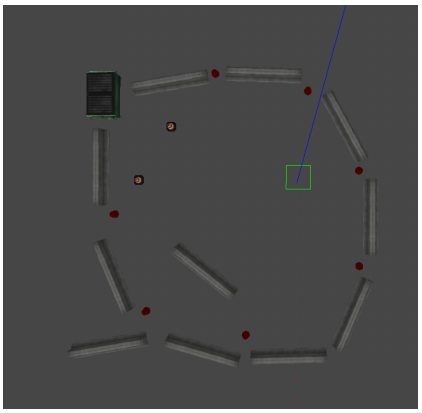
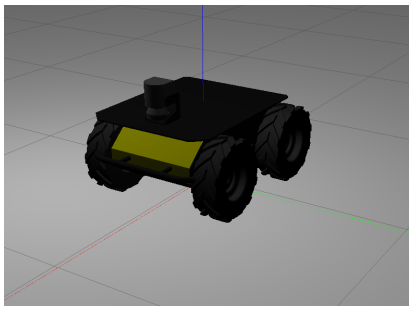


Figure 4.1: Images showing the simulated environment used for testing



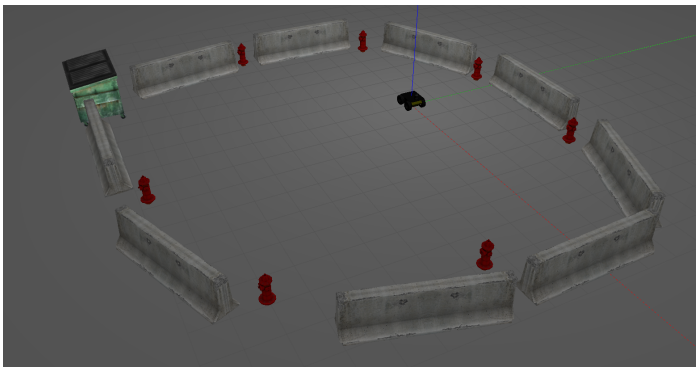
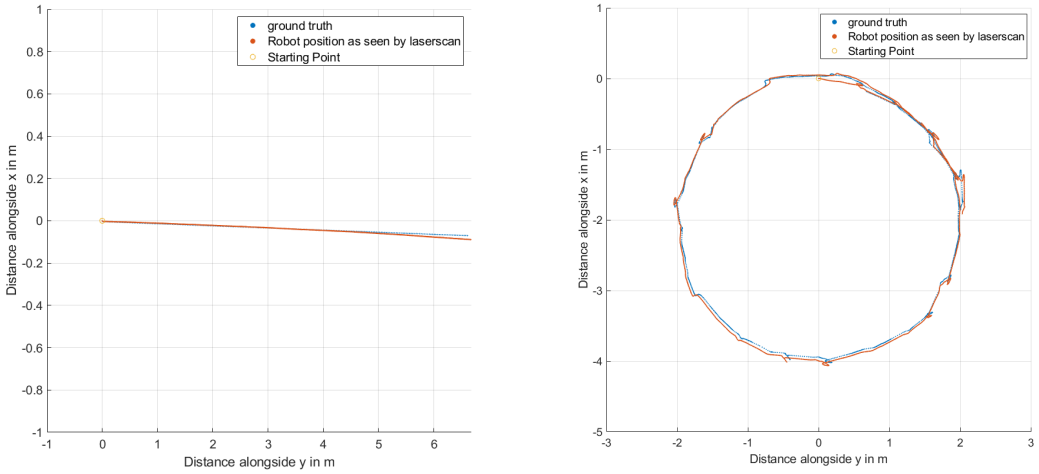


Figure 4.2: Images the husky robot in simulation (left) and its presence in the simulated environment



(a) Results for the robot driving in a straight line (no path following controller was used)

(b) Results for the robot going in a circular path

Figure 4.3: Images showing the position estimate of the robot given by the laser scan matcher.

4.1.2 Laser Scan tool accuracy evaluation

To evaluate the accuracy of the Laser Scan tool accuracy, we performed tests inside the Gazebo environment. We provided a perfect environment for the laser scan, the approximation of the position of the laser estimated by the laser scan was compared with the position of the robot given as a ground truth by the Gazebo software.

The results can be seen in figure 4.3 the figure on the left, the estimator of the laser scan is used to estimate the position of the robot while going in a straight line. It allows visualizing the results of the laser scan without being affected by our algorithm. Figure 2, presents a laser scan result of the deployment of our algorithm compared to the ground truth value obtained from the simulated environment. The estimation of the laser scan falls within an accuracy range inferior to 10 cm, placing it at the same accuracy level of an RTK-GPS.

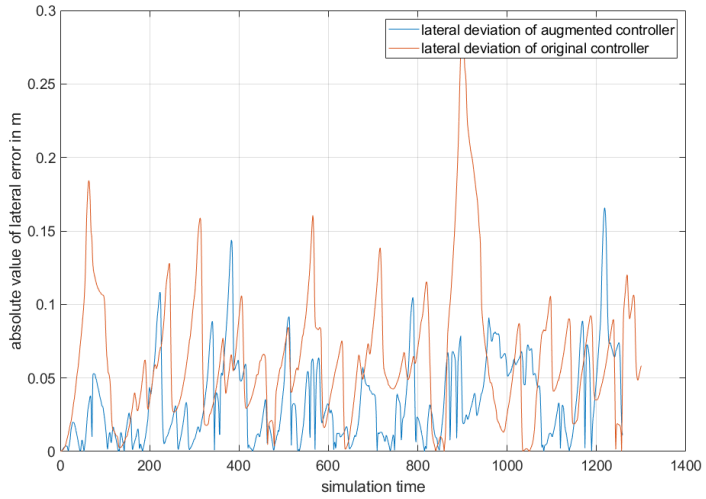


Figure 4.4: Lateral error of original controller vs the augmented one

4.1.3 Assessment and evaluation of Simulations

Preliminary testing was performed within the same Gazebo environment to validate our work and evaluate the effect of the augmented controller term seen in 2.18,

As it's seen, the augmented controller present more robustness in general. 4.4 showcases the true impact of the augmented controller. The augmented controller adds robustness. But otherwise, it does not improve the results drastically. Figure 4.5, 4.6 showcases the averages values of the lateral deviation for both controller under different slopes and different friction coefficients μ respectively . Each scenario was tested five times, in each of these 5 times the average deviation over time is kept. Each of the final values is, therefore, an average of the 5 values.

Figure 4.6 shows that for a higher coefficient, the lateral deviation is less important. This is expected since a higher coefficient means less slippage. The different values used for μ are 0.6, 0.85, 1, 1.2 and 10. To be noted, 10 is not a realistic value, however, it was added to showcase the behavior of the controllers in a slippage-less situation. The results of both controllers are not as sensitive to change in slope as much as it is sensitive to change is friction coefficient. To be noted, the augmented controller does showcase a more consistent

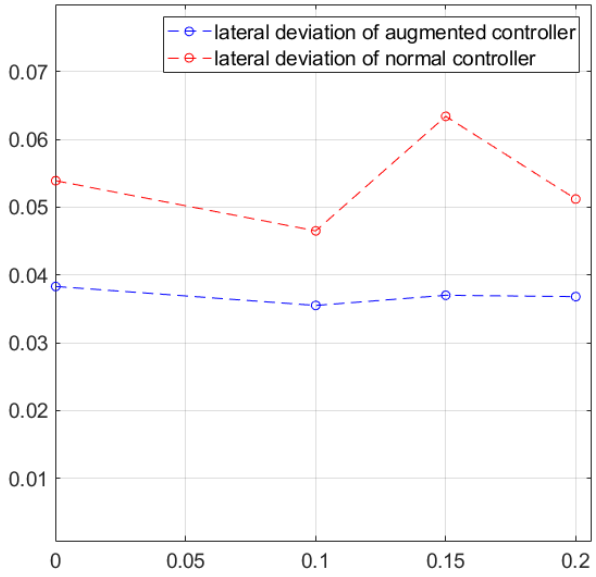


Figure 4.5: Lateral deviations of both controller with different ground slope

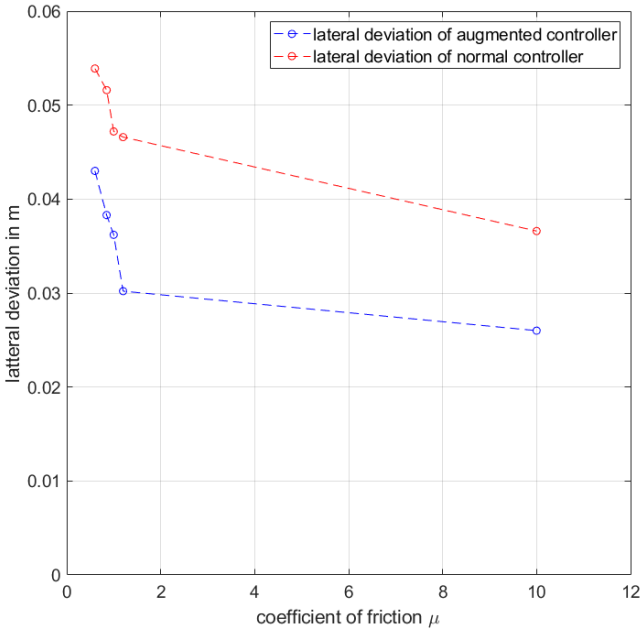


Figure 4.6: Lateral deviations of both controllers with different values of the friction coefficient μ

performance.

4.2 Testing Results

To validate our results, several live tests were performed. The test involved the robot operating autonomously in an indoor environment and a natural outdoor environment based solely on the computation done by the observer. These tests aimed to evaluate the sensitivity of our system to the nature of the ground (and hence the amount of slippage) and the number of objects present around it (which affects the functioning of the laser scan matcher tool and thus the odometry assessment).

We performed three tests, one test indoor, and two tests outdoor. The indoor testing was aimed to be a validation one since it provided a perfect space for the laser scan matcher to make sure it does not fail.

4.2.1 Indoor Testing

Indoor testings were used as a validation test given that the indoor environment provides a perfect space for the laser scan to work. Some images of the space are provided in figure 4.7. As we can see the environment is bordered by walls, adding to this we have several objects, the couches (in dark grey), the posters, and the different indoor trees (in green) that act as features for our laser scan. A schematic of this seen can be seen in figure 4.8. With the red doted line describing the trajectory followed by the robot. The schematic is just provided for visualization purposes and it is not to scale.

Figures 4.9, 4.10, and 4.11 showcases respectively in the indoor test; the position of the robot and desired path, the lateral deviation values, as well the approximation of the angle β. Two runs are shown, as it is visible in figures 4.9 and 4.10 the robot is capable of following the robot with a good accuracy keeping the lateral deviation under 5 cm. Figure 4.11 showcases that the value of β is quite constant and minimal.

(a) Lateral error values (b) position of the robot with the trajectory

Figure 4.7: Figure showcasing images of the indoor space used for testing

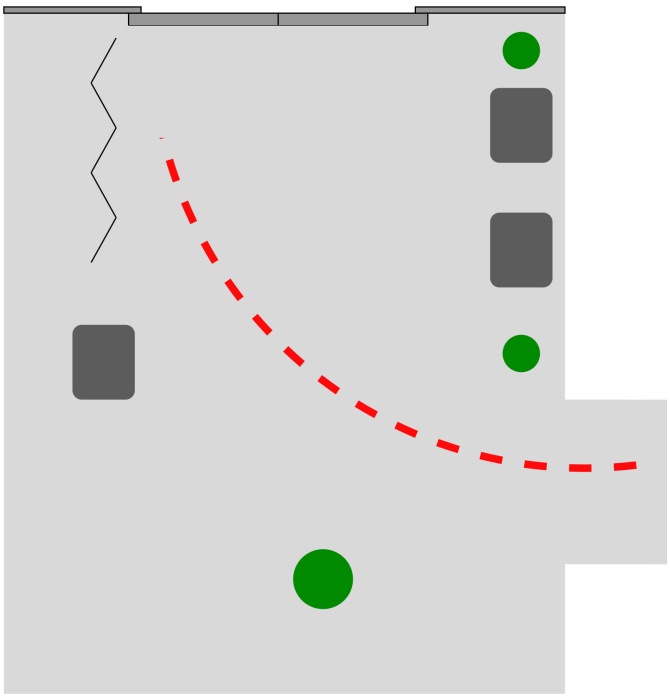


Figure 4.8: Schematic map of the indoor environment

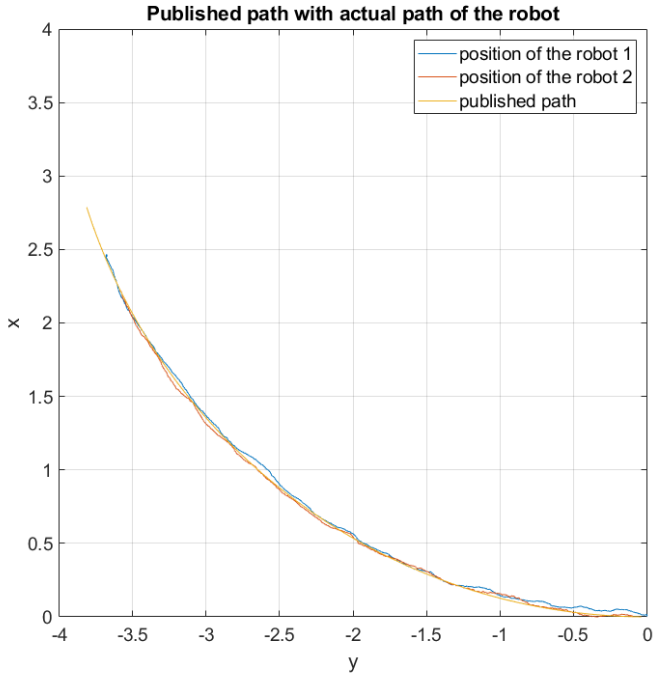


Figure 4.9: Figures showing position of the robot, compared with the desired path of the robot, in the indoor testing environment

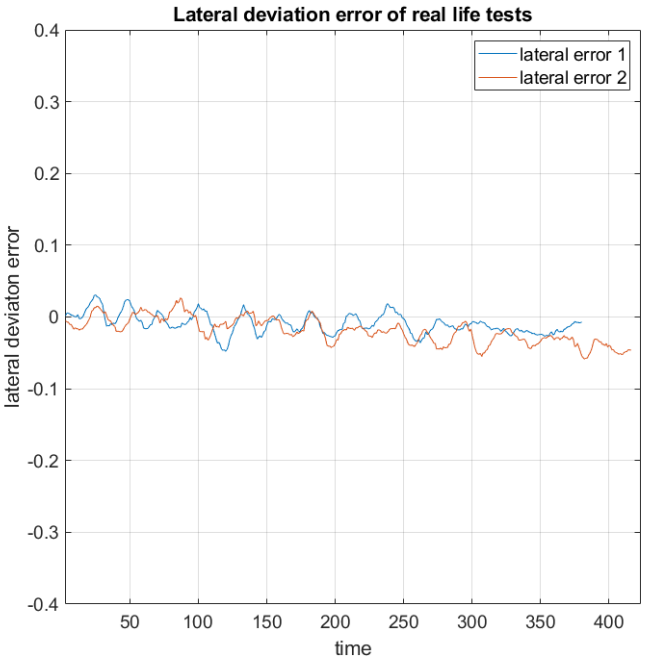


Figure 4.10: Figures showing the lateral deviation values and the position of the robot in the indoor testing locations

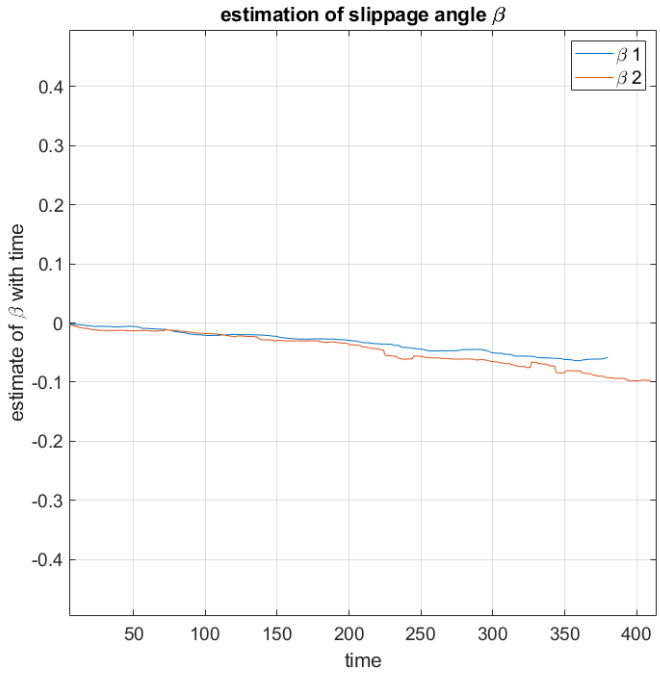


Figure 4.11: Figures showing the lateral deviation values and the position of the robot in the indoor testing locations

4.2.2 Outdoor Testing

Finally, to validate the performance of the system in a natural environment several tests were performed in two different locations. Figure 4.12 showcases a bird-eye view of the campus, given that 2 the solution was tested in 2 different environments, the figure was labeled with 1 and 2 to show the location of these environments on campus.

Environment 1

A schematic of the environment 1 is seen in figure 4.13. The dark grey presents the campus main building, the red figures are meant to symbolize the different car parked nearby, the dark green is vegetation, and the object brown is a bench. The light green area showcases the vegetation on which the testing was done.

As seen in figure 4.13, this testing environment provided enough features for the laser scan matcher to function properly. The system did not face any problem until it became

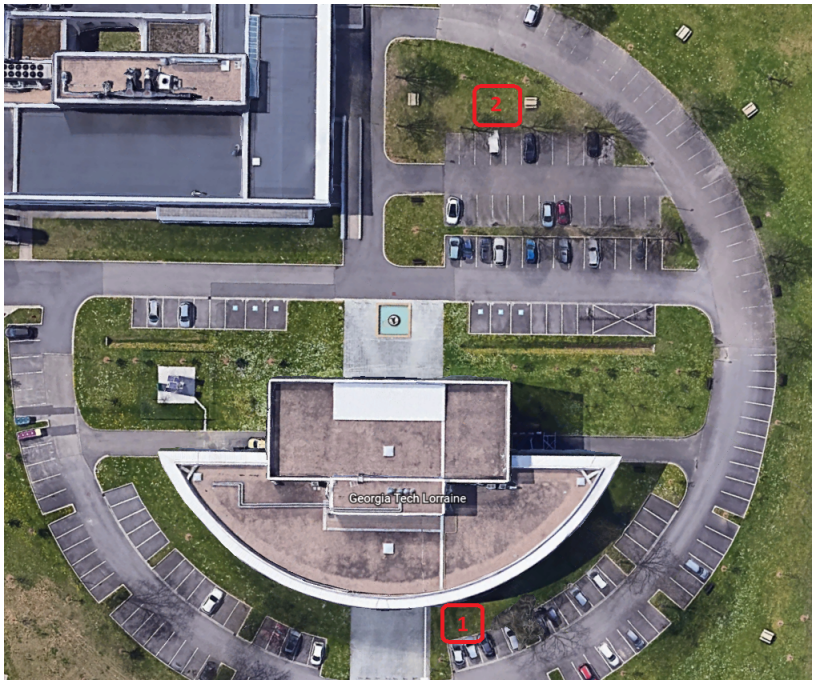


Figure 4.12: figure showcasing a bird-eye view of the campus with the testing environments labeled as 1 and 2

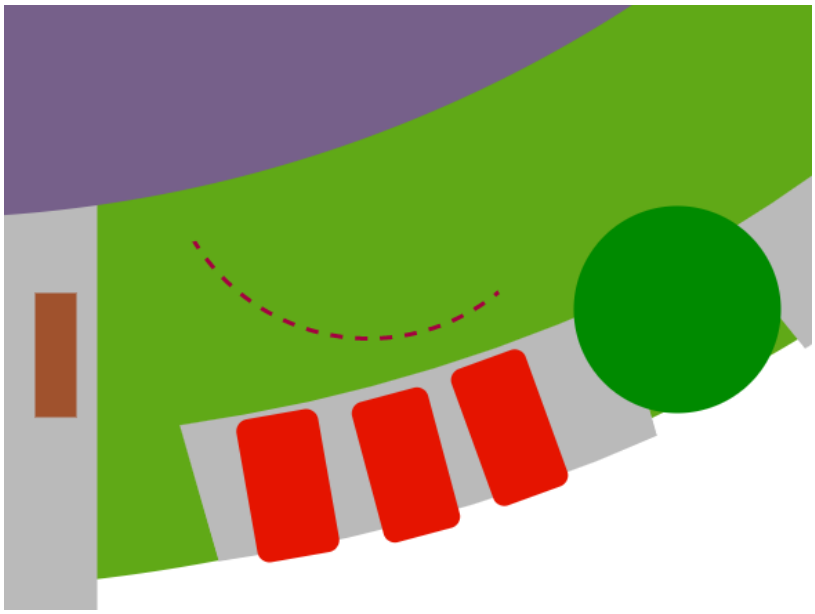


Figure 4.13: Schematic map of natural environment testing location 1

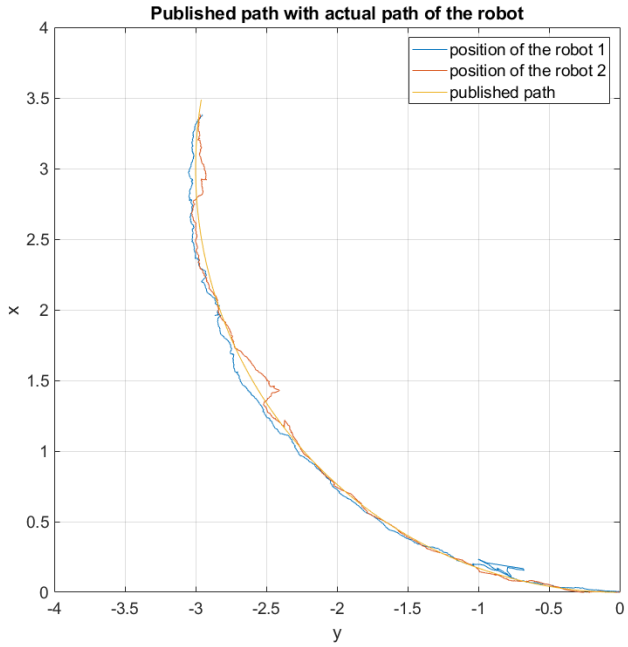


Figure 4.14: position of the robots with respect to the desired path in testing location 1

directly in front of the building. In fact, given the convex shape of the building the laser scan matcher cannot resolve the different points properly.

Figures 4.14,4.15, and 4.16 showcases the results of the tests in this environment, our system performs positively as we can see. The lateral deviation of the robot is below 10 cm with a short exception with the orange path, reviewing the details of the recorded message this is associated with a laser scan not functioning properly. This is actually seen in 4.14, at the coordinates $(-2.5; -1.5)$, the robot exhibits a lateral movement, an action impossible for a skid-steer robot.

It is interesting to see that this behavior is coupled with a radical value of β in figure 4.16 with β jumping from 3 to -3. This only makes sense to the estimator since it would approximate an extremely large lateral slippage as an explanation of this scenario. While the is the obvious laser mismatch, the value of the β of the blue path suggests two other mismatches, one of them that corresponds to the first peak in the β value can be attributed to the strange behavior the blue path exhibits towards the beginning at the coordinates

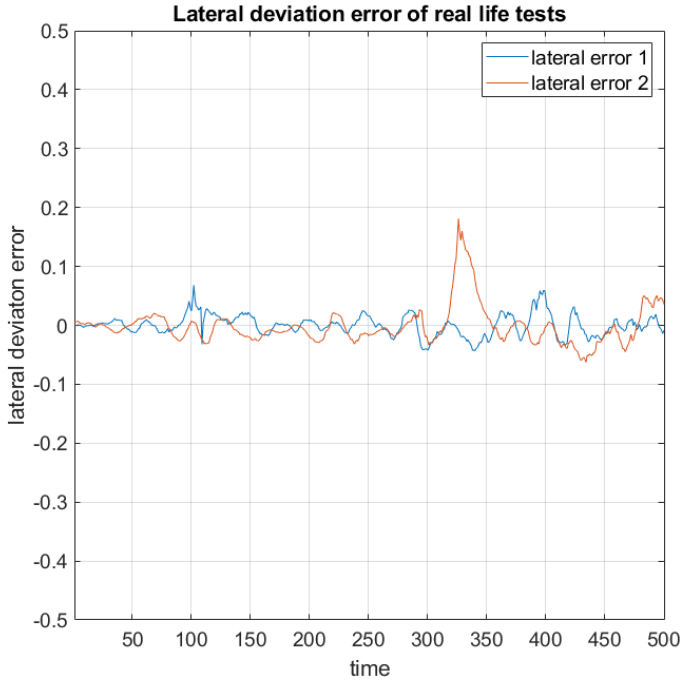


Figure 4.15: Figures showing the lateral deviation values in the outdoor testing location 1

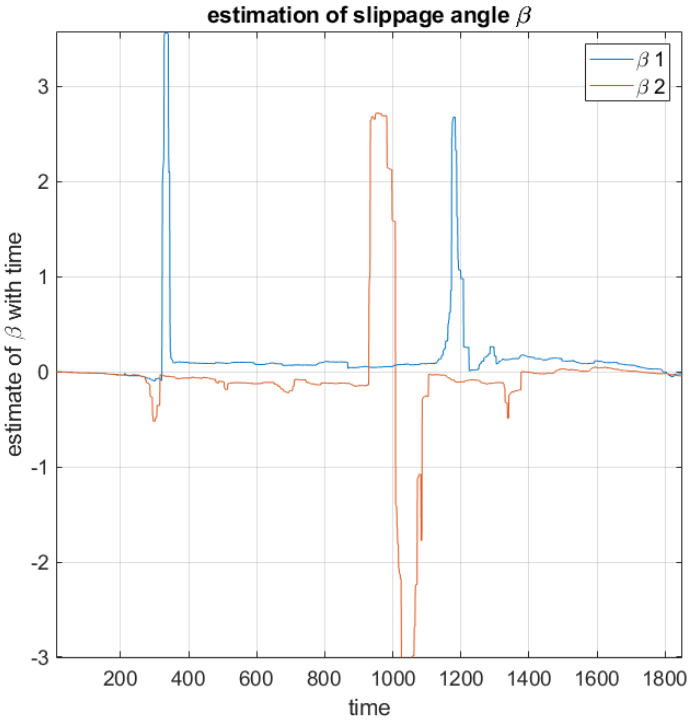


Figure 4.16: Figures showing the estimated values of the slippage angle β the outdoor testing location 1

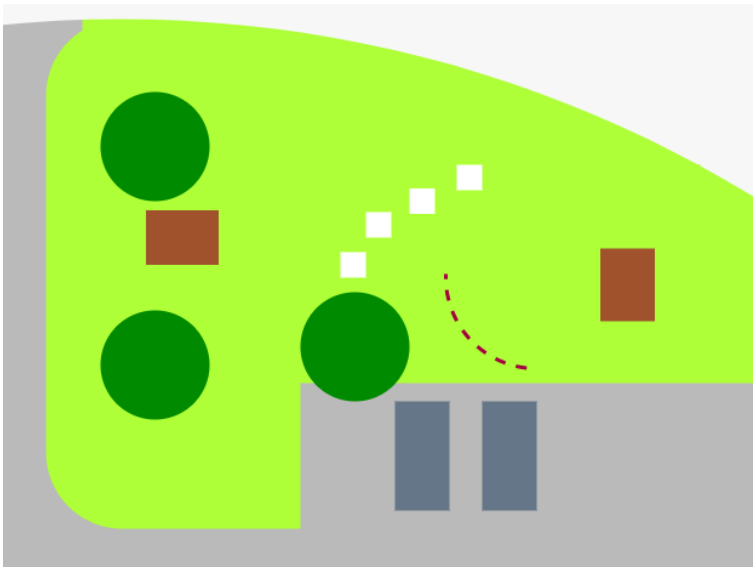


Figure 4.17: Schematic map of natural environment testing location 2

$(-1; -0.7)$. Hence, in some cases, the model can overcome laser mismatch problems.

Environment 2

Similarly, a schematic of the environment 2 is seen in figure 4.17. The dark grey presents the cars nearby, the dark green is vegetation, and the object brown is a bench and picnic tables. The light green area showcases the vegetation on which the testing was done.

We note the presence of small white boxes. In fact, in our preliminary testing, the laser scanning matcher was failing often in this environment. Hence, we decided to add some objects to the environment to guarantee successful runs of the experiment.

A very important note about the second environment, the tests were carried after a rainy day, hence the ground was muddy. Looking at figures 4.18 and 4.19, the path trajectory algorithm is functioning properly even in this environment.However, we do note a certain failure of the blue trajectory after some point. Looking at the value of β for the blue trajectory we can see that after some point the values stop making sense. Looking at the βs values in figure 4.20, we notice that this time the values exhibit faster changes when compared to the first test. This makes sense, given the muddy nature of the ground the

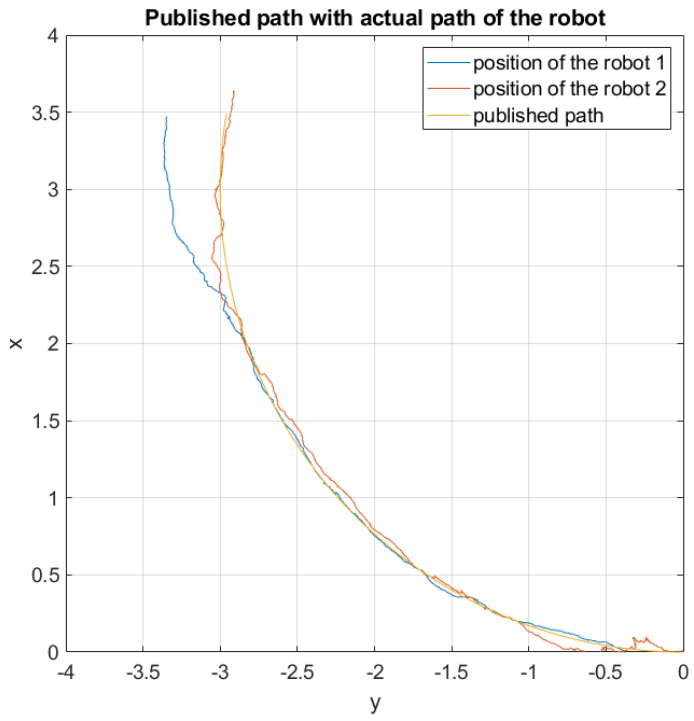


Figure 4.18: position of the robots with respect to the desired path in testing location 2

slippage experienced is highly non-uniform.

Figure 4.19: Figures showing the lateral deviation values in the outdoor testing location 2

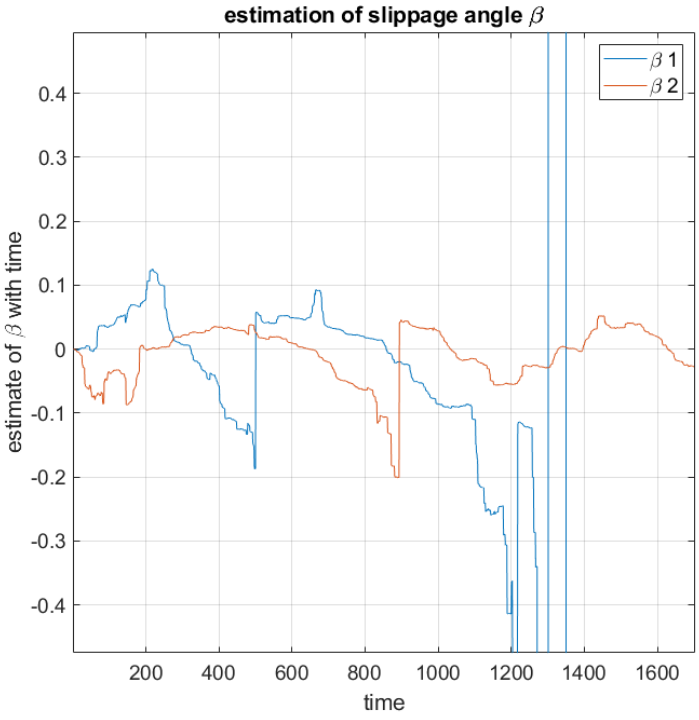


Figure 4.20: Figures showing the estimated values of the slippage angle β the outdoor testing location q

CHAPTER 5
DISCUSSION & CONCLUSION

The document at hand presented a possible solution to trajectory following under slippage-caused uncertainty. The work-based itself on an exteroceptive sensor, that is a 2D laser scan to estimate the position and the movement of the robot in the environment. The slippage of the robot was estimated using a traditional observer technique.

It was proven that a controller based on a back-stepping model with some modifications can provide a satisfactory result for the slippage compensation of a skid-steer robot, even without taking into consideration a proper model of the robot.

The results of our experiments are satisfactory if the environment around the robot provides enough features for the laser odometry to function. The results are highly dependent on the environment around the robot. It was seen in chapter4 that the experimental results are very satisfactory indoor and in simulation, while they deteriorate in the outside environment based on the quantities of features present for the laser scan matcher.

In the following sections, we will be discussing possible solutions that could resolve some of the limitations we had:

5.1 Using a more accurate robot model

It was seen that the equation used to model the robot did not match exactly with its physical model. This choice was made to make things simpler and to make the entire system transferable given that the differential drive robot is the simplest model there is.

One first possible improvement on this would be to use the equations of the skid-steer robot as seen in [1], or at least something closer to it than what is used. A more correct model would be able to grasp the reality much more than the one used.

Another possible improvement also in the modeling part, would be the possibility of

implementing a low-level controller. As seen in figure 2.2, the robot control system consists of a high-level controller and a low-level one. If an improved model was adopted, this would allow us to compute individual command for each of the wheels allowing us, therefore, to control each wheel on its own. However, this is not possible in the case of the Husky robot since each of the 2 wheels are chain-linked and controlled by the same motor.

5.2 A better perception model

Another possible path for improvement is the perception model used for the robot. In our solution we adopted a 2D Laser Scan matcher, better tools can be used including:

5.2.1 Jump rejection in scan matcher

We noted several times that there is a mismatch in the scan matcher that causes the robot to "jump" in space, moving instantaneously in a lateral manner. This cause the value for β to explode, as seen in 4.16 and 4.20. A solution to this would be to modify the scan matcher in such a way it rejects these jumps and to make sure the robot movement approximation is consistent with the physical properties of the robot.

5.2.2 3D laser scan

A 3D laser scan sensor and its matching counterpart would provide a more robust matching and therefore a more robust odometry estimation. In fact, given that a 3D laser scan has more than 1 array to do its matching makes it more robust. To be noted, I would still prefer a 3D laser scan to an RGBD camera for several reasons, the main being, dealing with the outdoor environment would be very challenging for an image matcher. Also, the added benefit of using a camera does not justify the added computational cost required for it. A 2D laser scan only detects objects at a certain height. Hence, even there are enough objects around the robot, an approach based on a 2D laser scan might fail where a 3D one won't.

Figure 5.1 showcases a simplistic description of the difference between 2D and 3D

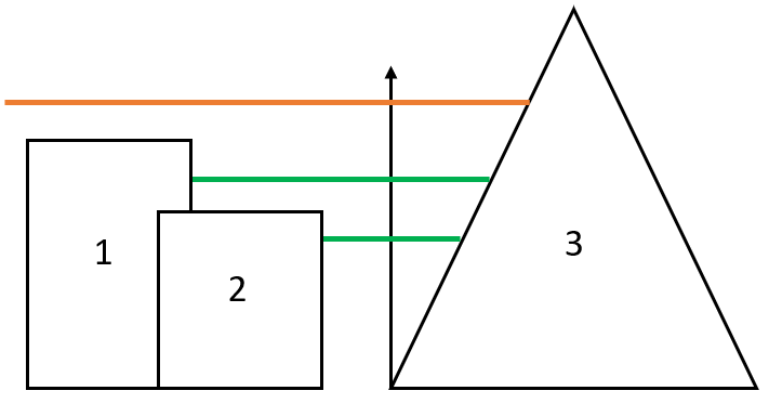


Figure 5.1: Illustration of resolution of 2D vs 3D lasers

lasers. The 2D laser can resolve only the object on one plane, while a 3D laser scan can detect the object on multiple scales. Figure 5.1 presents a cross-cut parallel to the (x, z) plane, with the arrow pointing in the positive z direction. If the robot is the arrow, a 2D laser scan would resolve only the object present on the orange plane. While a 3D laser would be able to detect all objects on the orange and the green planes. Thus, in simple terms, a 3D laser has more features to match.

In fact, during our testing, we had some problems with a sloppy area. Given the nature of the 2D scan, the scan matcher was catching a section of the slope that was identical across the entire scan. The scan matcher had a group of points on a certain distance d of the robot, and while it was moving forward the scan matcher was still seeing a group of points at a distance d, the scan matcher thus assumes the robot is not moving.

5.2.3 A sensor fusion technique

Another possible solution would be to augment the results of the laser scan with a more global sensor such as a GPS sensor. That would make the system capable of relying on the GPS for its global positioning and the laser scan for its accurate local positioning as well as giving it the capacity of ”relocating itself” in case of a laser scan matching error. A high-level architecture of this possible implementation is seen in figure 5.2.

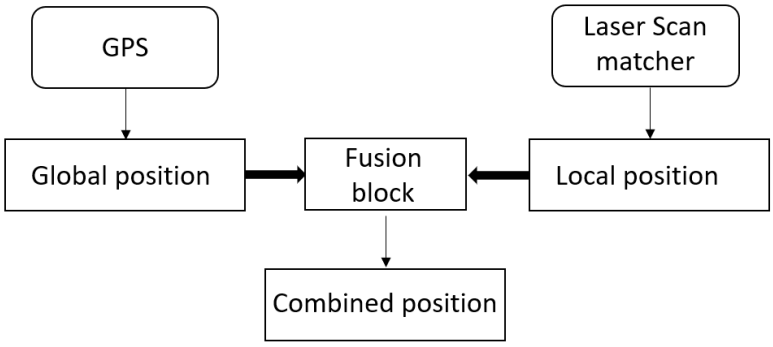


Figure 5.2: A possible sensor fusion implementation

REFERENCES

[1] T. Wang, Y. Wu, J. Liang, C. Han, J. Chen, and Q. Zhao, "Analysis and experimental kinematics of a skid-steering wheeled robot based on a laser scanner sensor," *Sensors*, vol. 15, no. 5, pp. 9681–9702, Apr. 2015.

[2] Y. Kanayama, Y. Kimura, F. Miyazaki, and T. Noguchi, "A stable tracking control method for an autonomous mobile robot," in *Proceedings., IEEE International Conference on Robotics and Automation*, 1990, 384–389 vol.1.

[3] Sung-On Lee, Young-Jo Cho, Myung Hwang-Bo, Bum-Jae You, and Sang-Rok Oh, "A stable target-tracking control for unicycle mobile robots," in *Proceedings. 2000 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2000) (Cat. No.00CH37113)*, vol. 3, 2000, 1822–1827 vol.3.

[4] Z.-P. JIANGdagger and H. NIJMEIJER, "Tracking control of mobile robots: A case study in backstepping**this paper was not presented at any ifac meeting. this paper was recommended for publication in revised form by associate editor alberto isidori under the direction of editor tamer başar.," *Automatica*, vol. 33, no. 7, pp. 1393 – 1399, 1997.

[5] J. G. Iossaqui and J. F. Camino, "Wheeled robot slip compensation for trajectory tracking control problem with time-varying reference input," in *9th International Workshop on Robot Motion and Control*, 2013, pp. 167–173.

[6] X. Li, Zhuping Wang, Xiaozhen Chen, J. Zhu, and Qijun Chen, "Adaptive control of unicycle-type mobile robots with longitudinal slippage," in *2016 American Control Conference (ACC)*, 2016, pp. 1637–1642.

[7] M. Yue, L. Wang, and T. Ma, "Neural network based terminal sliding mode control for wmrs affected by an augmented ground friction with slippage effect," *IEEE/CAA Journal of Automatica Sinica*, vol. 4, no. 3, pp. 498–506, 2017.

[8] H. Gao, X. Song, L. Ding, K. Xia, N. Li, and Z. Deng, "Adaptive motion control of wheeled mobile robot with unknown slippage," *International Journal of Control*, vol. 87, no. 8, pp. 1513–1522, 2014. eprint: `https://doi.org/10.1080/00207179.2013.878038`.

[9] M. Deremetz, R. Lenain, B. Thuilot, and V. Rousseau, "Adaptive trajectory control of off-road mobile robots: A multi-model observer approach," in *2017 IEEE International Conference on Robotics and Automation (ICRA)*, 2017, pp. 4407–4413.

[10] M. Deremetz, R. Lenain, A. Couvent, C. Cariou, and B. Thuilot, "Path tracking of a four-wheel steering mobile robot: A robust off-road parallel steering strategy," in *2017 European Conference on Mobile Robots (ECMR)*, 2017, pp. 1–7.

[11] M. Deremetz, "Contribution to the modelling and control of autonomous and adaptable mobile robots in natural environments," Theses, Université Clermont Auvergne, Jul. 2018.

[12] R. Mur-Artal and J. D. Tardós, "ORB-SLAM2: An open-source SLAM system for monocular, stereo and RGB-D cameras," *IEEE Transactions on Robotics*, vol. 33, no. 5, pp. 1255–1262, 2017.

[13] P. McGarey, F. Pomerleau, and T. Barfoot, "System design of a tethered robotic explorer (TReX) for 3D mapping of steep terrain and harsh environments," in *Proceedings of the Conference on Field and Service Robotics (FSR). Springer Tracts in Advanced Robotics*, vol. 113, 2016, pp. 267–281.

[14] F. Pomerleau, F. Colas, R. Siegwart, and S. Magnenat, "Comparing ICP variants on real-world data sets: Open-source library and experimental protocol," *Autonomous Robots*, vol. 34, no. 3, pp. 133–148, 2013.

[15] A. Censi, "An icp variant using a point-to-line metric," Jun. 2008, pp. 19 –25, ISBN: 978-1-4244-1646-2.

[16] M. Smith, I. Baldwin, W. Churchill, R. Paul, and P. Newman, "The new college vision and laser data set," *The International Journal of Robotics Research*, vol. 28, no. 5, pp. 595–599, May 2009.

[17] S. Sastry, *Nonlinear Systems*. Springer New York, 1999.

[18] C. Samson, "Path following and time-varying feedback stabilization of a wheeled mobile robot," *Second International Conference on Automation, Robotics and Computer Vision*, vol. 3, Jan. 1992.